7

A NeuroGenetic Approach for Multiprocessor Scheduling

Anurag Agarwal
Department of Information Systems and Operations Management, Warrington College of Business Administration, University of Florida
USA

1. Abstract

This chapter presents a NeuroGenetic approach for solving a family of multiprocessor scheduling problems. We address primarily the Job-Shop scheduling problem, one of the hardest of the various scheduling problems. We propose a new approach, the NeuroGenetic approach, which is a hybrid metaheuristic that combines augmented-neural-networks (AugNN) and genetic algorithms-based search methods. The AugNN approach is a non-deterministic iterative local-search method which combines the benefits of a heuristic search and iterative neural-network search. Genetic algorithms based search is particularly good at global search. An interleaved approach between AugNN and GA combines the advantages of local search and global search, thus providing improved solutions compared to AugNN or GA search alone. We discuss the encoding and decoding schemes for switching between GA and AugNN approaches to allow interleaving. The purpose of this study is to empirically test the extent of improvement obtained by using the interleaved hybrid approach instead of applied using a single approach on the job-shop scheduling problem. We also describe the AugNN formulation and a Genetic Algorithm approach for the Job-Shop problem. We present the results of AugNN, GA and the NeuroGentic approach on some benchmark job-shop scheduling problems.

2. Introduction

Multiprocessor scheduling problems occur whenever manufacturing or computing operations are to be scheduled on multiple machines, processors or resources. A variety of such scheduling problems are discussed in the literature. The most general scheduling problem is the resource-constrained project-scheduling problem; this problem has received a lot of attention in the literature Herroelen et al. (1998), Kolisch (1996). The open-shop, flow-shop, job-shop and task scheduling problems can be considered special cases of the resource-constrained project-scheduling problem. While smaller instances of the various types of scheduling problems can be solved to optimality in reasonable computing time using exact solution methods such as branch and bound, most real-world problems are unsolvable in reasonable time using exact methods due to the combinatorial explosion of the feasible solution space. For this reason, heuristics and metaheuristics are frequently employed to obtain satisfactory solutions to these problems in reasonable time. In this

Source: Multiprocessor Scheduling: Theory and Applications, Book edited by Eugene Levner,
ISBN 978-3-902613-02-8, pp.436, December 2007, Itech Education and Publishing, Vienna, Austria

paper, we propose a new hybrid metaheuristic approach called the NeuroGenetic approach for solving one family of multiprocessor scheduling problems – the job-shop scheduling problem. The NeuroGenetic approach is a hybrid of the Augmented Neural Networks (AugNN) approach and the Genetic Algorithms (GA) approach. The AugNN approach provides a mechanism for local search, while the GA approach provides a mechanism for global search. An interleaving of the two approaches helps guide the search to better solutions.

In this chapter, we focus on the job-shop scheduling problem (JSSP). In JSSP, there are n jobs, each having m operations and each operation requires a different machine, so there are m machines. For each job, the order in which operations require machines is fixed and is independent of the order of machine requirement on other jobs. So in a 2x3 job shop-problem, for example, say job 1 requires machines in the order 2, 3 and 1, job 2 may require the machines in a different order, say 1, 3 and 2 or 1, 2 and 3 or 3, 1 and 2 or it could be the same i.e., 2,3 and 1. In a flow-shop problem (FSP), which is special case of the job-shop problem, the order in which machines are needed for each operation is assumed to be the same for each job. An FSP is therefore, a special case of the JSSP. In both JSSP and the FSP, there is only one machine of each type, and a machine can only process one operation at a time. The problem is to find a precedence and resource feasible schedule for each operation for each job with the shorted possible makespan. In general, preemption is not allowed, i.e. operations must proceed to completion once started.

A job-shop scheduling problem can be considered a special case of the resource-constrained project scheduling problem (RCPSP). In the RCPSP, a PERT chart of activities can be drawn just like for a JSSP. The RCPSP is more general because it allows multiple successors for an operation, whereas a JSSP allows only one successor. Also, while in RCPSP an activity may require multiple units of multiple resource types, in JSSP activities require only one unit of one resource type. Task scheduling problem is also a special case of RCPSP, in that only one type of resource is required for all activities. In task scheduling there can be multiple successors for an operation, like in an RCPSP.

In the next section, we review the literature primarily on JSSP. In the following section, the AugNN formulation for a JSSP is described. Section 4 outlines the GA approach for solving the JSSP. Section 5 describes the Neurogenetic approach and discusses how the AugNN and GA approaches can be interleaved. Section 6 provides the computational results of several benchmark problems in the literature. Section 7 summarizes the paper and offers suggestions for future research. This study contributes to the literature of job-shop scheduling by proposing for the first time an AugNN architecture and formulation for the JSSP and also proposing a hybrid of AugNN and GA approach.

3 Literature Review

The JSSP has been recognized as an academic problem for over four decades now. Giffler and Thompson (1960) and Fisher and Thompson (1963) were amongst the first to address this problem. Exact solution methods have been proposed by Carlier and Pinson (1989), Applegate and Cook (1991) and Brucker et al. (1994). A number of heuristic search methods have also been proposed, for example, Adams et al. (1988) and Applegate and Cook (1991). A variety of metaheuristic approaches have also been applied to the JSSP, such as Neural Networks (Sabuncuoglu and Gurgun, 1996), Beam Search (Sabuncuoglu and Bayiz, 1999), Simulated Annealing (Steinhofel et al. 1999), Tabu Search (Barnes and Chambers, 1995;

Nowicki and Smutnicki, 1996; Pezzella and Merelli, 2000; Zhang et al. 2008), Genetic Algorithms (Falkenauer and Bouffoix, 1991; Storer et al, 1995; Aarts et al., 1994; Bean, 1994; Croce et al., 1995), Evolutionary Algorithms (Mesghouni and Hammadi, 2004), Variable Neighborhood Search (Sevkli and Aydin, 2007), Global Equilibrium Search technique (Pardalos and Shylo, 2006). Jain and Meeran (1999) provide a good survey of techniques used for the JSSP. For the RCPSP, a number of heuristic and metaheuristic approaches have been proposed in the literature. For a good review of the heuristics, see Herroelen et al., 1998.

4. Augmented Neural Network Formulation

The AugNN approach was first introduced by Agarwal et al. (2003). They applied the AugNN approach to the task scheduling problem and offered an improved approach for using AugNN approach in Agarwal et al. (2006). In this approach, a given scheduling problem is converted into a neural network, with input layer, hidden layers and output layer of neurons or processing elements (PEs). The connections between the PEs of these layers are assigned weights. Input, activation and output functions are then designed for each node in such a way that a single-pass or iteration from the input to the output layer produces a feasible solution using a heuristic. An iteration, or a pass, consists of calculating all the functions of the network from the input up to the output layer. A search strategy is then applied to modify the weights on the connections such that subsequent iterations produce neighboring solutions in the search space.

We now describe, with the help of an example, how to convert a given JSSP problem into a neural network. We will assume a simple 3x2 JSSP instance of Figure 1 for this purpose.

Job →	**1**	**2**	**3**
Machine (Proc Time)	1 (4)	2 (5)	1 (3)
	2 (3)	1 (4)	2 (6)

Figure 1. An Example 3x2 Job Shop Scheduling Problem

In this 3x2 problem, there are 3 jobs, each with 2 operations, for a total of 6 operations (O_{11}, O_{12}, O_{21}, O_{22}, O_{31} *and* O_{32}). Job 1 requires 4 units of time (ut) on machine 1 (O_{11}) followed by 3 ut on machine 2 (O_{12}). Job 2 requires 5 ut on machine 2 (O_{21}) followed by 4 ut on machine 1 (O_{22}). Job 3 requires 3 ut on machine 1 (O_{31}) followed by 6 ut on machine 2 (O_{32}). The problem is how to schedule these six operations such that the makespan is minimized. Figure 2 shows a neural network for this problem.

There are two operation layers, corresponding to the two operations for each job. Each operation layer has three nodes corresponding to each job. Note that for a more general nxm case, there will be m operation layers, each with n nodes. Following each operation layer is a machine layer with 3 nodes each. Each of the three operation nodes is connected to a machine which is determined by the given problem. So, for example, given our 3x2 problem of Figure 1, O_{11} is connected to machine 1 and O_{12} is connected to machine 2; O_{21} is connected to machine 2 and O_{22} is connected to machine 1, and so on. For a more general n x m case, there will be n machine nodes in each of the m machine layers. An input layer is designed to provide a signal to the first operation layer to start the scheduling process. There is also an output layer with one PE labeled O_F for "final operation", which is a dummy operation with zero processing time and no resource requirement. The operation and the machine layers can be regarded as

hidden layers of a neural network. Connections between operation nodes and machine nodes are characterized by weights. These weights are all the same for the first iteration, but are modified for subsequent iterations. There are also connections between machine nodes and subsequent operation nodes, which are not characterized by any weights. These connections serve to pass signals from one layer to the next to trigger some functions.

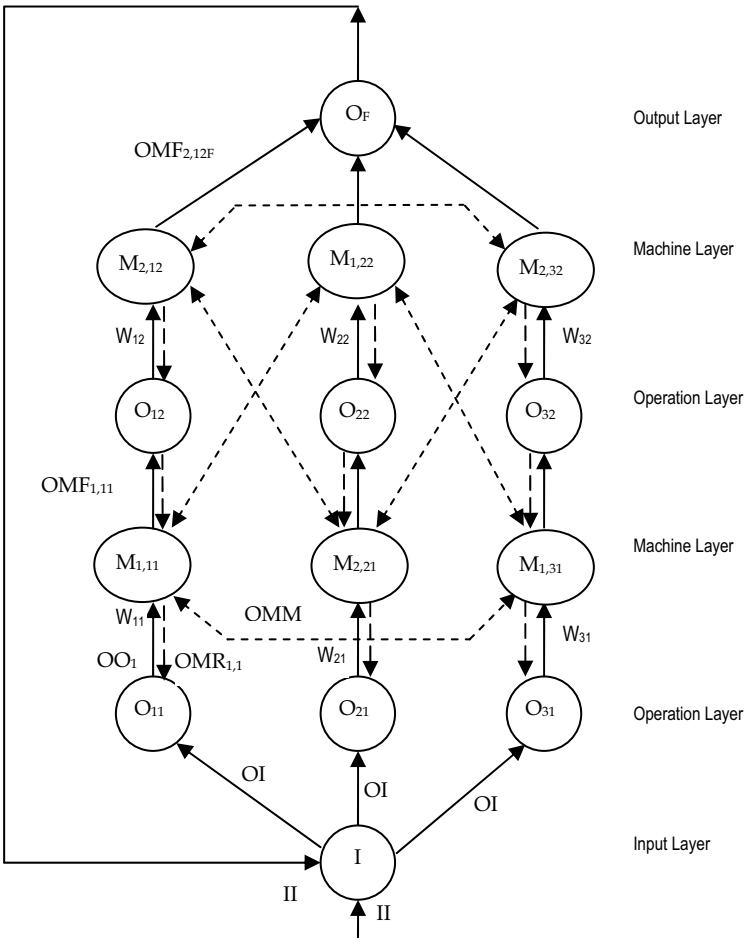


Figure 2: AugNN Architecture to solve a 3x2 Job Shop Scheduling Problem

The output of the operation nodes (OO) becomes input to the machine nodes. There are three types of outputs from each machine node. One output (OMF) goes to the next operation node (or to the final node). This signals the end of an operation on that machine. The second type of output (OMM) goes to the machine node of its own type. For example, machine 1 sends an output to all other machine 1 nodes. Similarly, machine 2 sends an

output to all other machine 2 nodes. These signals are used to enforce the constraint that the same machine cannot process more than one operation at the same time. The third output (OMR) is in the reverse direction, back to the operation node. Whenever an operation is assigned to a machine, the machine node sends a signal back to the operation node, indicating that it has been assigned. This signal changes the state of the operation node and triggers other functions.

We now describe the input, activation and output functions for each node and the search strategy for the weights. We will need the following notation to describe our functions:

n	Number of jobs
m	Number of machines
C	Current iteration
J	Set of jobs = {1,..,n}
J_i	Job i, $i \in J$
M	Set of machines = {1,..,m}
M_k	Machine k, $k \in M$
O	Set of operations
O_{ij}	ij^{th} operation node, $i \in J$, $j \in M$
$M_{k,ij}$	Node for machine k, connected from O_{ij}, $i \in J$, $j \in M$, $k \in M$
ω_{ij}	Weight on the link from O_{ij} to machine node, $i \in J$, $j \in M$
ω_m	Large weight on the link between machine nodes.
α	Search coefficient
ε_c	Error in iteration c
O_F	Final Dummy operation node
ST_{ijk}	Start time of O_{ij} on M_k, $i \in J$, $j \in M$, $k \in M$
PT_{ij}	Processing Time of J_i on M_j, $i \in J$, $j \in M$
Win_{ij}	Winning status of Job J_i on Machine M_j, $i \in J$, $j \in M$

Following are all functions of elapsed time t :

t	Elapsed time
$II(t)$	Input function value of the Initial I node.
$IO_{ij}(t)$	Input function value of Operation node O_{ij}, $i \in J$, $j \in M$
$IO_F(t)$	Input function value of Operation node O_F
$IM_{k,ij}(t)$	Input function value of Machine node k from operation O_{ij}, $i \in J$, $j \in M$, $k \in M$
$OI(t)$	Output function value of the Initial I node.
$OO_{ij}(t)$	Output function value of Operation node O_{ij}, $i \in J$, $j \in M$
$OO_F(t)$	Output function value of Operation node O_F
$OMF_{k,ij}(t)$	Output of Mc. node $M_{k,ij}$ to the operation node in forward direction, $i \in J$, $j \in M$, $j \neq m$, $k \in M$
$OMF_{k,ijF}(t)$	Output of Machine node $M_{k,ij}$ to O_F in the forward direction, $i \in J$, $j=m$, $k \in M$
$OMR_{k,ij}(t)$	Output of Machine node $M_{k,ij}$ to O_{ij} in reverse direction, $i \in J$, $j \in M$, $k \in M$
$OMM_k(t)$	Output of Machine node M_{k^*} to M_{k^*} $k \in M$
$\theta O_{ij}(t)$	Activation function of Operation node O_{ij}, $i \in J$, $j \in M$
$\theta M_{k,ij}(t)$	Activation function of Machine node $M_{k,ij}$, $i \in J$, $j \in M$, $k \in M$
$assign_{ijk}(t)$	Operation O_{ij} assigned to Machine M_k
$S(t)$	Set of operations that can start at time t. $S(t) = \{O_{ij} \mid OO_{ij}(t) = 1\}$

The neural network algorithm can be described with the help of the input, activation and output functions for the various PEs (input node, operation nodes, machine nodes and the final node) and the search strategy.

- **AugNN Functions**

Input Layer (Node I):
Input function: $II(0) = 1$
Output function: $OI(0) = II(0)$
Operation Layer Nodes:
Input function:

$$IO_{i1}(0) = II(0) = 1, \ \forall\ i \in J \tag{1}$$

$$IO_{ij}(0) = 0\ , \ \forall i \in\ J, j \in M, j > 1 \tag{2}$$

$$IO_F(0) = 0 \tag{3}$$

These functions at time t = 0 provide initial signals to the operation layers. The first operation nodes of all the jobs (i.e. for $j = 1$) get a starting signal of 1 at time 0 (equation 1). The remaining operation layers get a signal of 0 (equation 2) and the final output layer also gets a signal of 0 (equation 3).
For time t > 0, we have the following functions:
For all other operations i.e. $\forall\ j > 1 \wedge t > 0$

$$IO_{ij}(t) = IO_{ij}(t\text{-}1) + \ OMF_{k,ij-1}(t)\ , \ \forall i \in J, j \in M, j > 1, k \in M \tag{4}$$

$$IO_F(t) = IO_F(t\text{-}1) + \sum OMF_{k,ijF}(t)\ , j=m\ , \ \forall k \in M, i \in J \tag{5}$$

IO_{ij} (equation 4) helps to enforce the constraint that a new operation of a job cannot start unless the current operation is completed. At $t = 0$, IO_{ij} is 0. When an operation node gets a signal from the machine node (*OMF*, described later), IO_{ij} becomes 1, which indicates that it is ready to start.
IO_F (equation 5) is the input of the final node. It gets an input from all the machines nodes of all the jobs. When IO_F becomes n, we know that all jobs are done.
Activation function:
Operation nodes' initial activation state (i.e. at t=0) is 1.
$\forall\ i \in J,\ j \in M,$

$$\theta O_{ij}(t) = \begin{cases} 1 & \text{if} \quad IO_{ij}(t) = 0 \\ 2 & \text{if} \quad (\theta O_{ij}(t\text{-}1) = 1 \vee 2) \wedge IO_{ij}(t) = 1 \\ 3 & \text{if} \quad (\theta O_{ij}(t\text{-}1) = 2 \vee 3) \wedge OMR_{k,ij}(t) = -1 \\ 4 & \text{if} \quad \theta O_{ij}(t\text{-}1) = 4 \vee (\theta O_{ij}(t\text{-}1) = 3 \wedge OMR_{k,ij}(t) = 0) \end{cases}$$

State 1 above implies that operation O_{ij} is not ready to be assigned because input to this operation is still 0. State 2 implies that the operation is ready to be assigned to a machine

because its input is 1. State 3 implies that the operation is in process because it is receiving a negative signal from a machine *k* that it is currently being processed. State 4 implies that the operation is complete and the negative signal from machine *k* is no longer there.
Output functions:

$$OO_{ij}(t) = \begin{cases} 1 & \text{if } \theta O_{ij}(t) = 2 \quad \forall\ i \in J, j \in M \\ 0 & \text{otherwise} \end{cases}$$

If an operation is ready to start (i.e. $\theta O_{ij}(t) = 2$), then the operation node sends a unit signal to the machine node that it can be assigned.
Machine Layer Nodes:
Input function:

$$IM_{k,ij}(t) = OO_{ij}(t) * \omega_{ij} + \sum OMM_k(t) * \omega_m \qquad \forall\ i \in J, j \in M, k \in M \quad (6)$$

There are two components of $IM_{k,ij}(t)$. The first component ($OO_{ij}(t) * \omega_{ik}$)is the weighted output from operation node O_{ij}. Whenever it is positive, it means that machine *k* is being requested by operation O_{ij} for assignment. Remember that OO_{ij} becomes 1 whenever it is ready to be assigned. The second component is either zero or large negative. The second component becomes large negative whenever machine *k* is already busy with another operation.
Activation function:

$$assign_{ijk}(t) = \begin{cases} 1 & \text{if } IM_{k,ij}(t) * HeuristicParameter > 0 \\ & \qquad \forall i \in J, j \in M, k \in M \\ 0 & \text{otherwise} \end{cases}$$

We have mentioned earlier that the AugNN functions, in addition to enforcing the constraints of the problem, also help embed a chosen heuristic into the problem. We have also seen how using the output of the operation node, The assignment takes place if the product of input of the machine node and the heuristic dependent parameter, (such as Processing Time or Earliest Finish Time) is positive and highest. The requirement for it being positive is to honor the inhibitory signals. The requirement for highest is what enforces the chosen heuristic.

If $assign_{ijk}(t) = 1$, then $ST_{ijk} = t$. Whenever an assignment takes place, we record the start time of the operation O_{ij} on machine *k*

If $|S(t)| > 1$ then if $assign_{ijk}(t) = 1$ then $Win_{ik} = 1$
The Win_{ik} term will be used later during the search strategy. We want to modify the weights of links based on whether a particular operation node won the competition in case there was more than one node competing for assignment.
Machine nodes' Initial Activation State (i.e. at t=0) is 1.

$\forall\, i \in J, j \in M,\ k \in M,$

$$\theta M_{k,ij}(t) = \begin{cases} 1 & \text{: machine available} \\ 2 \text{ if } (\theta M_{k,ij}(t\text{-}1) = 1 \vee \theta M_{k,ij}(t) = 1) \wedge assign_{ijk}(t) = 1 & \text{:machine busy (just assigned)} \\ 3 \text{ if } (\theta M_{k,ij}(t\text{-}1) = 2 \vee 3) \wedge t < ST_{ijk} + PT_{ik} & \text{: machine busy (processing)} \\ 4 \text{ if } \theta M_{k,ij}(t\text{-}1) = 3 \wedge t = ST_{ijk} + PT_{ik} & \text{: machine just finished processing} \\ 5 \text{ if } \theta M_{k,ij}(t\text{-}1) = 1 \wedge \sum OMM_k(t) * \omega_m < 0 & \text{: assigned to another job} \\ 6 \text{ if } \theta M_{k,ij}(t\text{-}1) = 4 & \text{: machine } k \text{ is finished processing } O_{ij} \\ 1 \text{ if } (\theta M_{k,ij}(t\text{-}1) = 1 \vee 5) \wedge \sum OMM_k(t) * \omega_m = 0 & \text{: released by other job or not assigned to any other job} \\ 1 \text{ if } \theta M_{k,ij}(t\text{-}1) = 1 \wedge \sum OMM_k(t) * \omega_m < 0 & \text{: available but operation assigned} \end{cases}$$

At t = 0, all machines are available (State 1). When an assignment occurs on a machine, that machine enters state 2 (Busy, just assigned). State 2 turns into state 3 (Busy) the following time unit and state 3 continues till the machine is processing an operation. As soon as a machine is done processing it enters state 4 (Just finished). When a particular machine node is assigned to an operation, all other machine nodes that represent the same machine enter state 5. For example, if machine node $M_{1,11}$ is assigned to operation O_{11} then machine nodes $M_{1,31}$, $M_{1,22}$ also enter state 5. In state 5, they cannot be assigned to another operation. When a machine is finished processing an operation, it reaches state 6 (Just released). A machine node enters the state of 1 from a state of 5 if it stops receiving a negative signal from other machine nodes.

Output functions:

$$OMF_{k,ij}(t) = \begin{cases} 1 & \text{if } \theta M_{k,ij}(t) = 4 \\ 0 & \text{if } \theta M_{k,ij}(t) = 1,2,3,5,6, \end{cases} \quad \forall\, i \in J,\ j,k \in M$$

Whenever a machine node is done processing an operation, i.e. it reaches state 4, it sends a signal to the operation ahead of it that it may start.

$$OMR_{k,ij}(t) = \begin{cases} -1 & \text{if } \theta M_{k,ij}(t) = 2,3 \\ 0 & \text{if } \theta M_{k,ij}(t) = 1,4,5,6, \end{cases} \quad \forall\, i \in J,\ j,\ k \in M$$

Whenever a machine node is busy processing an operation (i.e. in states 2 or 3), it sends a negative signal to the operation node that it is processing. This helps switch the state of the operation node from 2 to a 3.

$$OMM_k(t) = \begin{cases} 1 & \text{if } \theta M_{k,ij}(t) = 2,3 \\ 0 & \text{if } \theta M_{k,ij}(t) = 1,4,5,6, \end{cases} \quad \forall\, i \in J,\ j,\ k \in M$$

Whenever a machine node is busy processing an operation (i.e. in states 2 or 3), it also sends a signal to other machine nodes corresponding to the same machine in other machine layers. This ensures that the same machine is not assigned to another job at the same time.

Output Layer (Node F)

The output of F represents the makespan and the $assign_{ijk}(t)$ gives the schedule. If a machine is either assigned or released during a certain time unit, all functions need to be recalculated without incrementing the time clock.

Input function:

$$IO_F(t) = IO_F(t\text{-}1) + OMF_{k,ijF}(t)$$

Output function:

$$OO_F(t) = \begin{cases} t & \text{if} \quad IO_F(t) = n \\ 0 & \text{otherwise} \end{cases}$$

The final node outputs the makespan (t), the moment it receives n signals (one from each job) indicating that all jobs are complete.

Search Strategy:

A search strategy is required to modify the weights. The idea behind weight modification is that if the error is too high, then the probability of a different machine being the winner should be higher during subsequent iteration. Since the machine with the highest value of *IM*, is the winner, an increase of weights will make the machine more likely to win and conversely a decrease of weight will make it less likely. The magnitude of change should be a function of the magnitude of the error and of some job parameter, such as processing time. Keeping these points in mind, the following search strategy is used for the weights on the links.

Winning tasks: If $Win_{ik} = 1$ then

$$(\omega_{ik})_{c+1} = (\omega_{ik})_c - \alpha * PT_{ik} * \varepsilon_c \quad \forall\ i \in J,\ k \in M$$

Non-winning Tasks: If $Win_{ik} = 0$ then

$$(\omega_{ik})_{c+1} = (\omega_{ik})_c + \alpha * PT_{ik} * \varepsilon_c \quad \forall\ i \in J,\ k \in M$$

When the above functions and search strategies are employed, each pass or iteration provides a feasible solution.

- **End of iteration routines:**

Calculate the gap (the difference between obtained makespan and the lower bound). Lower bound is the time of the critical path on the PERT chart, assuming infinite resources. The lower bound can be calculated once at the beginning.

1. Store the best solution so far.
2. If the lower bound is reached, or if the number of iterations is greater than a specified number, stop the program.
3. If continuing with the next iteration, modify weights using the search strategy.

5. Genetic Algorithm

Many different chromosome encodings have been suggested for the JSSP. For example, Falkenauer and Bouffouix (1991) proposed a chromosome formed of several subchromosomes, one for each machine; each subchromosome is a string of symbols, each symbol identifying an operation that has to be processed on the relevant machine. Croce et al. (1995) used the same encoding as Falkenauer and Bouffouix. Bean (1995) used a random key alphabet U(0,1), a vector of random numbers. Each solution chromosome is made of $2n$ genes where n is the number of operations. The first n genes are used as operation priorities, whereas the genes between $n+1$ and $2n$ are used to determine the delay times used when scheduling an operation. Dagli and Sittisathanchai (1995) use a chromosome with $n.m$ genes, where n is the number of jobs and m the number of machines, each gene represents an operation. The order of genes represents the order in which the operations will be scheduled.

In this work, we use the representation similar to the one used by Dagli and Sittisathanchai (1995), i.e. there will be $n.m$ number of genes, each gene represents an operation number, and the order of the genes dictates the order in which the operations are scheduled. Care has to be taken that the ordering of operations is feasible. Any order in which the operations of each job are in the required order would be a feasible ordering. The GA algorithm is described as:

{ Generate an initial population of feasible ordered chromosomes P_i, where $i = 1$
Evaluate each chromosome in the initial population.
While stopping criteria is not met, repeat
{ Select best chromosomes of initial population to copy to the next population. P_i+1
Crossover best chromosomes of Pi and place into Pi+1
Mutate chromosomes in P_i and place in P_i+1
Evaluate population P_i+1
}
}

Crossover

The crossover mechanism should be such that the resulting child chromosome must produce a feasible schedule. In other words, the priority order represented by the child chromosome must be precedence feasible. We use a two-point crossover scheme. In a two-point crossover, two integer points c_1 and c_2 are randomly generated such that $c_2 > c_1$ and $c_2 - c_1 > nm/3$ and both c_1 and c_2 are between 1 and $n.m$. Two parent chromosomes are used as input. The child chromosome genes are produced as follows:

Genes1 through c_1 from parent 1 go the child chromosome as is. Genes $c_1 + 1$ to c_2 genes of the child come from parent 2 using the rule that any unused genes in parent 2 starting from the first position are placed in the child till c_2 genes in the child are filled. The remaining genes in the child come from parent 1 i.e. all unused genes appear in the child in the same order as they appear in parent 1. This rule ensures the feasibility of the schedule generated by the child chromosome.

Mutation

With a certain mutation probability, a certain number of genes are moved in such a way that the schedule remains precedence feasible, i.e. the order of operations with respect to a particular job is not disturbed, but the order of jobs with respect to other jobs may be

disturbed. Since the order of jobs between jobs is independent of each other, such a move maintains the precedence order of operations.

Evaluation

A given chromosome, which basically represents the ordering of the operations, is evaluated by generating a schedule. We perform better of forward and backward scheduling and also perform double justification to make sure the best possible schedule is obtained for the given chromosome. A parallel schedule generation scheme was found to be better than a serial schedule generation scheme for the job-shop problem. The Smallest Latest Finish Time Operation First and the Highest Remaining Work Next heuristics gave the best results.

6. Neurogenetic Approach

In the NeuroGenetic approach, we interleave the search between AugNN and GA. For example, we may run x number of generations of GA, take the best chromosome so far and try to improve upon this solution in the local search space, using y number of iterations of the AugNN search. However, in order to switch from GA search mode to AugNN search mode, appropriate weight vector has to be determined. The weight vector should be such that in conjunction with a chosen heuristic, AugNN would produce the same schedule as the given GA schedule. Using this set of weights and the chosen heuristic, we run say y iterations using the AugNN approach. If better solutions are found during the AugNN search iterations, these solutions can replace the worst solutions in the most recent GA population. GA search can then resume for another x number of generations, and so on till some stopping criteria is met. The critical part of this interleaving mechanism is how to determine the set of weights that would allow AugNN to replicate a given GA solution. We next describe an algorithm to determine the weights using the heuristic Highest Remaining Work (RWK) Next. We start with a unit weight vector for all activities. We will call the chromosome that we want to achieve using the weights and the RWK Next heuristic the target chromosome and the starting chromosome the source chromosome.

Algorithm for Switching from GA Encoding to AugNN Encoding

Create a source chromosome based on non-increasing order of RWK.
Repeat until each gene in the source chromosome is at the same as position as in the target chrom.
{ Let w_a and w_b represent the weights corresponding to the out of place gene and the target position gene
*If (w_a*RWK_a > w_b*RWK_b and $position_a$ > $position_b$) and*
*Set w_a > w_b * (RWK_b/RWK_a) = 0.1 + w_b * (RWK_b/RWK_a)*
End If
*Rearrange the source chromosome based on non-increasing order of w*RWK*
}

Example: let us say we are scheduling activities in a PERT chart and we are using the heuristic of "Max Remaining Work Next". Suppose there are eight activities and the vector F of their Remaining Work is (19, 12, 14, 10, 9, 6, 5, 0). Assume a vector of weights w = (1,1,1,1,1,1,1,1). Assume that GA produces a string of (1, 2, 3, 5, 4, 6, 7, 8) which is our target vector. The source vector S, based on the vector F would be (1, 3, 2, 4, 5, 6, 7, 8). We notice that the gene at positions 2, 3, 4 and 5 in S are different from the target vector. To bring gene in position 2 in S to position 3,

So, set w2 = w3 * (RWK3/RWK2) + 0.1

Or w2 = 1 * (14/12) + 0.1 = 1.267

So the new w = (1.0, 1.267, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0) and the new w.F = (19, 15.2, 14, 10, 9, 6, 5, 0). The new ordering based on w.F = (1, 2, 3, 4, 5, 6, 7, 8). At this point, gene in position 4 is not in the same position as the target.
So we set w5 = w4*(RWK4/RWK5) + 0.1
Or w5 = 1*(10/9) + 0.1 = 1.211
So the new w = (1.0, 1.267, 1.0, 1.0, 1.211, 1.0, 1.0, 1.0) and the new w.F = (19, 15.2, 14, 10, 10.9, 6, 5, 0). The new ordering based on w.F = (1, 2, 3, 5, 4, 6, 7, 8) which is the target string.

Switching from AugNN Encoding to GA Encoding

Switching the encoding schemes from AugNN to GA is a relatively straightforward. Whatever ordering of operations is obtained using the product of the weight vector and the heuristic parameter becomes the ordering of genes in the GA chromosome.

Instance	Size	BKS[1]	Heuristic	AugNN	GA	NeuroGenetic	Dev. (%)
MT06	6x6	55	55	55	55	55	0.00
MT10	10x10	930	1051	980	965	950	2.15
MT20	20x10	1165	1265	1182	1191	1178	1.12
ABZ5	10x10	1234	1287	1249	1252	1245	0.89
ABZ6	10x10	943	986	952	961	945	0.21
ABZ7	20x15	656	721	711	702	672	2.44
ABZ8	20x15	665	736	699	698	680	2.25
ABZ9	20x15	679	739	718	715	685	0.88
ORB01	10x10	1059	1145	1072	1082	1063	0.38
ORB02	10x10	888	919	902	905	893	0.56
ORB03	10x10	1005	1110	1008	1110	1007	0.19
ORB04	10x10	1005	1071	1051	1060	1031	2.58
ORB05	10x10	887	959	895	899	894	0.78
ORB06	10x10	1010	1110	1053	1042	1036	2.57
ORB07	10x10	397	431	410	405	399	0.50
ORB08	10x10	889	1034	925	930	910	2.36
ORB09	10x10	934	978	945	952	934	0.00
ORB10	10x10	944	1028	978	990	961	1.80
Average							1.20

[1]Best Known Solution

Table 1. Makespan using different algorithms on some well-known benchmark problems

Instance	Size	BKS[1]	Heuristic	AugNN	GA	NeuroGenetic	Dev. (%)
LA01	10x5	666	666	666	666	666	0.00
LA02	10x5	655	677	655	670	655	0.00
LA03	10x5	597	636	617	607	599	0.33
LA04	10x5	590	619	607	609	592	0.34
LA05	10x5	593	593	593	593	593	0.00
LA06	15x5	926	926	926	926	926	0.00
LA07	15x5	890	890	890	890	890	0.00
LA08	15x5	863	863	863	863	863	0.00
LA09	15x5	951	951	951	951	951	0.00
LA10	15x5	958	958	958	958	958	0.00
LA11	20x5	1222	1222	1222	1222	1222	0.00
LA12	20x5	1039	1039	1039	1039	1039	0.00
LA13	20x5	1150	1150	1150	1150	1150	0.00
LA14	20x5	1292	1292	1292	1292	1292	0.00
LA15	20x5	1207	1207	1207	1207	1207	0.00
LA16	10x10	945	1010	981	965	950	0.53
LA17	10x10	784	817	793	788	784	0.00
LA18	10x10	848	909	869	852	848	0.00
LA19	10x10	842	899	875	844	842	0.00
LA20	10x10	902	951	927	922	910	0.88
LA21	15x10	1046	1162	1127	1085	1047	0.09
LA22	15x10	927	1034	982	950	936	0.97
LA23	15x10	1032	1072	1032	1032	1032	0.00
LA24	15x10	935	1025	979	982	957	2.35
LA25	15x10	977	1105	1031	1016	988	1.12
LA26	20x10	1218	1311	1236	1241	1222	0.32
LA27	20x10	1235	1345	1296	1265	1261	2.10
LA28	20x10	1216	1363	1281	1295	1236	1.64
LA29	20x10	1152	1228	1189	1178	1166	1.21
LA30	20x10	1355	1418	1382	1388	1368	0.96
LA31	30x10	1784	1784	1784	1784	1784	0.00
LA32	30x10	1850	1850	1850	1850	1850	0.00
LA33	30x10	1719	1719	1719	1719	1719	0.00
LA34	30x10	1721	1752	1735	1730	1721	0.00
LA35	30x10	1888	1898	1888	1890	1888	0.00
LA36	15x15	1268	1451	1368	1325	1305	2.92
LA37	15x15	1397	1550	1457	1498	1446	3.51
LA38	15x15	1196	1311	1247	1258	1223	2.26
LA39	15x15	1233	1335	1256	1272	1242	0.73
LA40	15x15	1222	1354	1285	1271	1251	2.37
Average							0.62%

[1]Best Known Solution

Table 2. Makespan using different algorithms on Lawrence benchmark problems

7. Computational Results

We show results for several benchmark datasets including three problems from Fisher and Thompson (1963), 40 problems from Lawrence et al. (1984), five problems from Adams et al. (1988) and ten ORB problems. Tables 1 and 2 summarize the results. We run the AugNN, the GA and the NeuroGenetic algorithm for 1000 unique solution iterations each. The results are not the best as found in the literature, but we did not run our algorithm for long periods of time. We were interested in seeing whether the interleaving of AugNN and GA resulted in any improvements. In general, we found that the interleaved approach gave some improvement. We provide the best known result in the BKS column, the result of dispatch rule heuristic in the heuristic column, followed by the AugNN, the GA and the NeuroGenetic results. The last column shows the percent deviation of the NeuroGenetic makespan with respect to the best known solution. For the Lawrence problems (Table 2), the average deviation across the 40 problems was 0.61%; for the other 18 benchmark problems (Table 1), the average deviation was 1.2%. The heuristic gave the optimum solution for 15 of the 40 Lawrence problems, AugNN provided the optimum solution for 17 problems and NeuroGenetic approach provided optimum solution for 21 problems.

8. Summary and Conclusions

In this study we combine two metaheuristic search techniques, the Augmented Neural Networks and Genetic Algorithms approach to create a hybrid metaheuristic called the NeuroGenetic approach. We apply this hybrid approach to a multiprocessor scheduling problem, the job-shop scheduling problem to test if the hybridization helps improve the solution. The hybridization of AugNN and GA is achieved by interleaving the two approaches. Since the GA approach is better at diversification or global search whereas AugNN is better at intensification or local search, the combination provides improved solutions than either GA or AugNN search with the same number of iterations. Computational results showed that such hybridization provided improvements in the solutions, than if each technique was used alone. Given the encouraging results, more research needs to be done in this area. Such hybrid techniques can be applied to other scheduling problems and also on the job shop scheduling problem by applying other GA approaches that have performed well in the literature. The AugNN technique can also be hybridized with other non GA techniques such Tabu Search and Simulated Annealing approaches, which tend to give good results for the job-shop scheduling problem.

9. References

Aarts, E.H.L., Laarhoven, P.J.M.V., Lenstra, J.K., Ulder, J.L.J., A Computational Study of Local Search Algorithms for Job Shop Scheduling, *ORSA Journal on Computing*, 1994, 6(2), 118-125.

Adams, J., Balas, E. and Zawack, D., The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science,* 1988, 34(3), 391-401.

Agarwal, A., Jacob, V. and Pirkul, H., An Improved Augmented Neural-Networks Approach for Scheduling Problems, *INFORMS Journal on Computing*, 2006, 18(1), 119-128.

Agarwal, A., Jacob. V., Pirkul, H., Augmented Neural Networks for Task Scheduling, *European Journal of Operational Research*, 2003, 151 (3), 481-502.

Applegate, D., and Cook, W., A Computational Study of the Job-Shop Scheduling Problem, *ORSA Journal on Computing*, 1991, 3(2), 149-156.

Barnes, J.W., and Chambers, J.B., Solving the Job Shop Scheduling Problem with Tabu Search, *IIE Transactions*, 1995, 27, 257-263.

Bean, J.C., "Genetics and Random Keys for Sequencing and Optimization, *ORSA Journal on Computing*, 1994, 6, 154-160.

Brucker, P., Jurisch, B. and Sievers, B., A Branch and Bound Algorithm for Job-Shop Scheduling Problem, *Discrete Applied Mathematics*, 1994, 49, 105-127.

Carlier, J. and Pinson, E., An Algorithm for Solving the Job Shop Problem, *Management Science*, 1989, 35, 164-176.

Croce, F.D., Tadei, R. and Volta, G. A Genetic Algorithm for the Job Shop Problem, *Computers and Operations Research*, 1995, 22(1), 15-24.

Dagli, C.H., and Sittisanthanchai, S., Genetic Neuro-Scheduler: A New Approach for Job Shop Scheduling, *International Journal of Production Economincs*, 1995, 41, 135-145.

Falkenauer, E. and Bouffoix, S., A Genetic Algorithm for Job Shop, Proceedings of the 1991 *IEEE International Conference on Robotics and Automation*, 1991.

Fisher, H. and Thompson, G.L., Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, in: *Industrial Scheduling*, J.F. Muth and G.L. Thompson (eds.), 1963, Prentice Hall, Englewood Cliffs, NJ, 225-251.

Giffler, B. and Thompson, G.L. Algorithms for Solving Production Scheduling Problems, *Operations Research* , 1960, 8(4), 487-503.

Herroelen W., Demeulemeester E., and De Reyck B., Resource- constrained project scheduling: A survey of recent developments, *Computers & Operations Research,* 1998, 25 (4), 279-302.

Jain, A.S., and Meeran, S., Deterministic Job Shop Scheduling: Past, Present and Future, *European Journal of Operational Research*, 1999, 113, 390-434.

Kolisch R., Efficient priority rule for the resource-constrained project scheduling problem, *Journal of Operations Management*, 1996, 14(3), 179–192.

Mesghouni, K., and Hammadi, S., Evolutionary Algorithms for Job Shop Scheduling, *International Journal of Applied Mathematics and Computer Science*, 2004, 14(1), 91-103.

Nowicki, E. and Smutnicki, C., A Fast Taboo Search Algorithm for the Job Shop Scheduling Problem, *Management Science*, 1996, 6, 108-117.

Pardalos, P. and Shylo, Oleg., An Algorithm for the Job Shop Scheduling Problem Based on Global Equilibrium Search Techniques, *Computational Management Science*, 2006, 3(4) 331-348.

Pezzella, F., and Merelli, E., A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem, *European Journal of Operational Research*, 2000, 120, 297-310.

Sabuncuoglu, I., and Gurgun, B., A Neural Network Model for Scheduling Problems, *European Journal of Operational Research*, 1996, 93, 288-299.

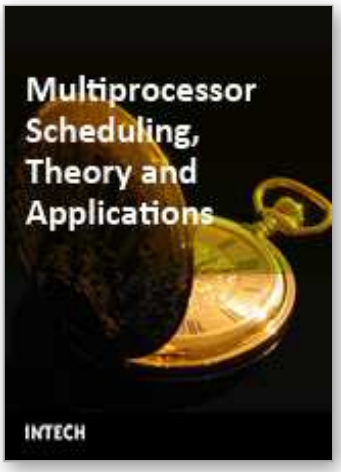
Sabuncuoglu, I., and Bayiz, M, Job Shop Scheduling with Beam Search, *European Journal of Operational Research*, 1999, 118, 390-412.

Sevkli, M., and Aydin, M.E., Parallel Variable Neigborhood Search Algorithms for Job Shop Scheduling Problems, *IMA Journal of Management and Mathematics*, 2007, 18, 117-133.

Steinhofel, K., Albrecht, A., and Wong, C.K., Two Simulated Annealing-Based Heuristics for the Job-Shop Scheduling Problem, *European Journal of Operational Research*, 1999, 118, 524-548.

Storer, R.H., Wu, S.D., and Vaccari, R., Problem and Heuristic Space Search Strategies for Job Shop Scheduling, *ORSA Journal on Computing*, 1995, 7(4) 453-487.

Zhang, C.Y., Li, P.G., Rao, Y.Q. and Guan, Z.L., A Very Fast TS/SA Algorithm for the Job Shop Scheduling Problem, *Computers & Operations Research*, 2008, 35, 282-294.



Multiprocessor Scheduling, Theory and Applications

Edited by Eugene Levner

ISBN 978-3-902613-02-8
Hard cover, 436 pages
Publisher I-Tech Education and Publishing
Published online 01, December, 2007
Published in print edition December, 2007

A major goal of the book is to continue a good tradition - to bring together reputable researchers from different countries in order to provide a comprehensive coverage of advanced and modern topics in scheduling not yet reflected by other books. The virtual consortium of the authors has been created by using electronic exchanges; it comprises 50 authors from 18 different countries who have submitted 23 contributions to this collective product. In this sense, the volume can be added to a bookshelf with similar collective publications in scheduling, started by Coffman (1976) and successfully continued by Chretienne et al. (1995), Gutin and Punnen (2002), and Leung (2004). This volume contains four major parts that cover the following directions: the state of the art in theory and algorithms for classical and non-standard scheduling problems; new exact optimization algorithms, approximation algorithms with performance guarantees, heuristics and metaheuristics; novel models and approaches to scheduling; and, last but least, several real-life applications and case studies.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821